CSE 2017 Data Structure and Lab
Midterm Exam, Spring 2019

Student Number: _____

Name: _____

Question 1[2 points each].
Decide whether this sentence is True or False. You should also give brief reasons for your answers.

(1) **True** **False** A return type can be specified on prototype for constructor in C++ programming language.

No return type can be specified on prototype for constructor. Constructor function behaves as if it returns object of its class type

(2) **True** **False** Changes in implementation of a class should not require changes in application that uses the class.

Hide the implementation details of the class from the users.

(3) **True** **False** The running time of InserItem() in unsorted list is O(N).

No. O(1)

.

(4) **True** **False** Time complexity of MakeEmpty() operation in Stack is same in both array and linked implementation.

No. O(1) in Array implementation, O(N) in linked implementaiton.

(5) **True** **False** Linear queue in array can have overflow even though it has enough room.

Yes. rear pointer can go to the max, but there can be space before front pointer.

(6) **True** **False** In a circular doubly linked list with 10 nodes, we will need to change 2 links if we want to delete a node other than the head node.

Yes. we just need to change 2 links if we want to delete a node other than the head node.

(7) **True** **False** A stack data structure should be used to keep track of the return addresses for nested functions while a program is running

Yes. The last calling function will be came back first, so stack should be used.

(8) **True** **False** If a circular queue is implemented as an array then elements are added to front position mod array size and removed from the back position.

Yes added to back position mod array size and removed from the front position.

(9) **True** **False** Time complexity of delete() operation in list are O(1) both case of arrays and linked-list implementation.

No, O(N) in array implementation, O(1) in linked list

(10) **True** **False** For implementing Insert position in sorted linked list two pointers are needed to keep previous trails behind location

Yes, using two pointers predLoc and location, where predLoc trails behind location

Question 2[10 points]

Based on the following declarations in C++ program, tell whether each statement below is syntactically legal (yes) or illegal(no).

```
int* p;
int* q;
int* r;
int a;
int b;
int c;
```

	yes/no
(1) `p = new int;`	no
(2) `q* = new int;`	yes
(3) `a = new int;`	yes
(4) `p = r;`	yes
(5) `q = b;`	no
(6) `r = NULL;`	yes
(7) `c = *q;`	yes
(8) `p = *a;`	no
(9) `delete b;`	no
(10) `q = &c;`	yes
(11) `delete r;`	yes
(12) `a= new p;`	no
(13) `q* = NULL;`	no
(14) `q* = a;`	yes
(15) `c = NULL;`	no

(2) What will be the output of the following code

```
#include <iostream>
#include <cstdlib>
using namespace std;
```

```
int main(){
   int d;
   int *p = (int *)operator new(sizeof(int));
   d = 5;
   *p = d;
   cout << ++*p  +  d++;
   return 0;
}
```

11(pointer not deleted after allocation with new)

Question 3[15 points].

Two stacks of positive integers are needed, one containing elements with values less than or equal to 1,000 and the other containing elements with larger than 1,000. The total number of elements in the small-value stack and the larger-value stack combined are not more than 200 at any time, but cannot predict how many are in each stack. [All of the elements could be in the small-value stack, they could be evenly divided, both stacks could be empty, and so on.] Can you think of a way to implement both stacks in one array?
(1) Draw a diagram of how the stack might look.

smallTop: top for the stack of small values, initialized to -1 and incremented.
largeTop: top for the stack of large values, initialized to 200 and decremented.

[0]	[1]	[2]	[3]	....	[197]	[198]	[199]

---

(2) Write the definitions for such a double-stack data structure(DStack).

```
class DStack {
public:
   void Push(int item);
   void PopLarge(int& item);
   void PopSmall(int& item);
private:
   int small;
   int large;
   int item[200];
}
```

(3) Implement the Push operation; it should store the new item into the correct stack according to its value(compared to 1,000).

```
void Push(int item) {
  if (item <= 1000) {
        small++;
        items[small] = item;
```

```
    }
    else {
          large--;
          items[large] = item;
    }
}
```

Question 4[10 points].

Please complete the following function to concatenate two circularly singly linked list into one circularly singly linked list.

```
typedef struct listNode * listPointer;
typedef struct listNode {
      int data;
      listPointer link;
};
```

```
listPointer concatenate(listPointer list1, listPointer list2) {
/* Produce a new list which contains list1 followed by list2.
Return the pointer which points to the new list.
The arguments list1 and list2 point to the tails of the lists.*/

  listPointer temp;
  if (list1==NULL)
        return list2;
  if (list2==NULL)
        return list1;
  listPointer tmp = list1->link;
  list1->link = list2->link;
  list2->link = tmp;
  return list2;


}
```

Question **5**[15 points]**.**

(1) Consider a stack of integers cStack. Using only the stack methods isEmpty(), pop(), pop(int x), and push(int y), write code (in C++ or pseudocode) which pops all the elements off cStack and returns the sum of all these elements

```
int sum = 0;
while (!cStack.isEmpty()){
    int x;
    cStack.pop(x);
    sum += x;
}
```

(2) How would the code above have to be modified if the stack cStack is to be left unchanged after the sum is found? (In your solution, still only use the stack public methods listed above.)

```
int sum = 0;
Stack<int> tempStack;
while (!cStack.isEmpty()){
        int x;
        cStack.pop(x);
        sum += x;
        tempStack.push(x);
}
while (!tempStack.isEmpty()){
        int x;
        tempStack.pop(x);
        cStack.push(x);
}
```

Question **6**[15 points]**.**

Trace though the state of the queue q in the following code fragment and show all status of queue by drawing. (Assume Queue is an implementation of the standard queue operations using C++ generic class. peek() returns just front element in queue)

```
Queue<Integer> q = new Queue<Integer>();
q.enqueue(5);
q.enqueue(7);
q.enqueue(13);
```

```
q.dequeue();
Integer t = q.peek();
q.enqueue(12+t);
q.dequeue();
q.enqueue(q.dequeue());
```

Solution:

q: (empty)
q: 5
q: 5 7
q: 5 7 13
q: 7 13
t:7
q: 7 13 19
q: 13 19
q: 19 13

Question **7**[15 points]..

Write a C++ Method to: Given a list of integers is stored in a Doubly Linked List (in no articular order). Given a pointer to the first node in the list, delete all occurrence of the node containing the integer x, and return a pointer to the first node.

public NodeType delAllOccurOfGivenKey (NodeType* headRef, int x)

```
template< class ItemType >
struct NodeType {
        int                         data;
        NodeType<ItemType>* back;
        NodeType<ItemType>* next;
};

class DoublyLinkedType {
public:
        DoublyLinkedType(int);
        ~DoublyLinkedType();
    void MakeEmpty();
    void FindItem(ItemType item, bool& found);
    void Replace(ItemType Newitem);
    bool IsEmpty() const;
    bool IsFull() const;
    void Insert(ItemType);
```

```
    void DeleteNode(listData, NodeType*current);   /* delete the node pointed to by  'current' */
    void Reverse();
private:
NodeType<ItemType> * listData, cursor;
int length;
}
```

```
void deleteAllOccurOfX(struct NodeType* head_ref, int x)
{
    /* if list is empty */
    if ((*head_ref) == NULL)
        return;

    struct NodeType* current = *head_ref;
    struct NodeType* next;

    /* traverse the list up to the end */
    while (current != NULL) {

        /* if node found with the value 'x' */
        if (current->data == x) {

            /* save current's next node in the
               pointer 'next' */
            next = current->next;

            deleteNode(head_ref, current);

            /* update current */
            current = next;
        }

        /* else simply move to the next node */
        else
            current = current->next;
    }
}
```